PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06K 15/02, G06F 3/12	A1	(11) International Publication Number: WO 99/24933 (43) International Publication Date: 20 May 1999 (20.05.99)

(21) International Application Number: PCT/BE98/00169

(22) International Filing Date: 5 November 1998 (05.11.98)

(30) Priority Data:
08/964,651 5 November 1997 (05.11.97) US

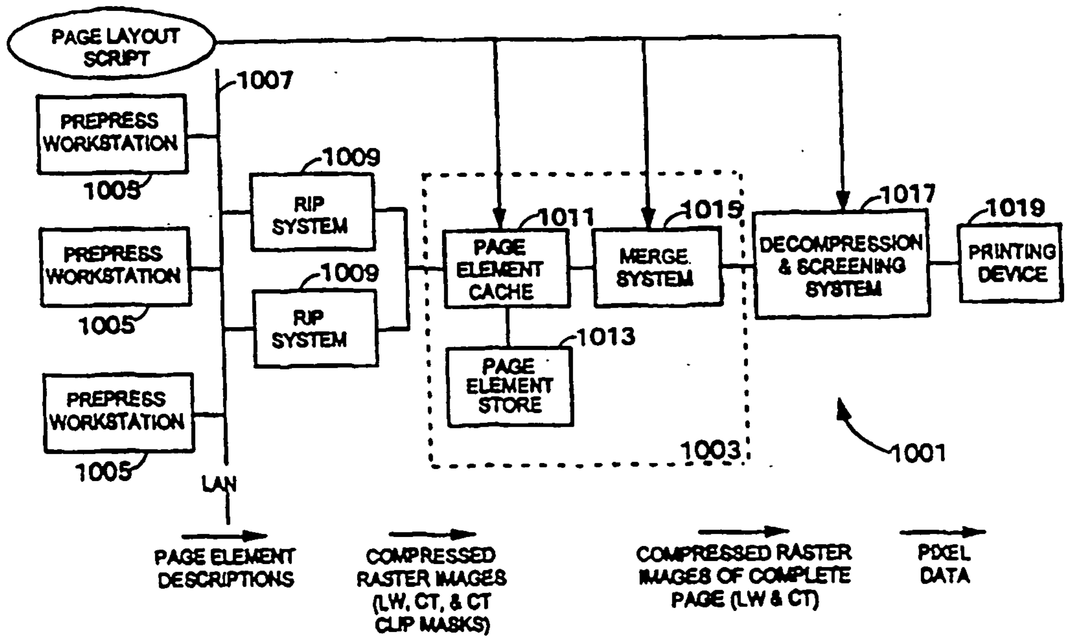
(71) Applicant: BARCO GRAPHICS N.V. [BE/BE]; Tramstraat 69, B-9052 Zwijnaarde (BE).

(72) Inventors: VLIETINCK, Jan, J.; Galglaan 9, B-9000 Gent (BE). NOTREDAME, Paul, H.; Klimoplaan 10, B-9032 Wondelgem (BE). DEBAERE, Eddy, H.; Oudestraat 7A, B-9800 Deinze (BE). DEPUYDT, Lieven, W.; Barco Patent Dept., Theodoor Sevenslaan 106, B-8500 Kortrijk (BE).

(81) Designated States: JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: MERGING OF COMPRESSED RASTER IMAGES IN A PRINTING SYSTEM



(57) Abstract

This invention relates to high speed digital printing of pages each obtained by merging one or more page elements. A method is disclosed for merging page elements which are stored in a compressed format, the merging substantially in compressed domain to enable the implementation of the method to perform the merge rapidly, keeping up with a fast printing device. The merging occurs according to a page layout script which specifies the positions and printing order of selected page elements on each page. An apparatus for merging also is disclosed.

MERGING OF COMPRESSED RASTER IMAGES IN A PRINTING SYSTEM

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to digital printing of pages containing variable information, with unrestricted variability from page to page. More particularly this invention relates to a method and an apparatus for merging a plurality of compressed data, the merging being carried out without decompressing the data.

Description of the Problem

Recent digital printing devices have made it possible to print page sequences where each printed page is completely different from the previous one. This is of particular importance in the production of personalized printed matter, *e.g.*, for direct mailing purposes. Such a system for printing personalized multi-page documents should have the following features:

- the possibility to store parts of those documents for reuse in other documents as page elements;
- an as good as unlimited number of pages in those documents (up to several hundreds of pages);
- no inherent restrictions to the amount of personalized data per page; and
- a high degree of freedom in the design of the personalization (a large number of regions, overlapping regions, *etc.*)

Such a printing system can be conceptually described as a page element based printing system: a document is created from pages, and each page is created from a number of elements stored in the system in a compressed raster form. In a typical workflow in which the present invention may be used, a prepress system produces the overall description of pages and of the graphical data elements which are the page elements.

A page is typically described using a page description language ("PDL") such as the PostScript® PDL and "PDF®" by Adobe Systems, Inc., Mountain View, California, HP PCL by Hewlett-Packard, Inc., Palo Alto, California, or by a format such as GRO™ ("GRaphics

those disks in a run length encoded ("RLE") compressed form. At print time one "master" background element can be combined with one "variable" foreground element with exactly the same bitmap size. Both elements are fully decompressed before merging.

The disadvantage of such a system is limited flexibility in the design of the personalized data: there cannot be multiple "variable" elements, covering different parts of multiple "master" elements on one page. This existing system also is not very scaleable, and it is not clear how well it could be adapted to drive faster print engines.

The Barco Graphics PrintStreamer I is an example of a restricted type of variable data system that has the concept of a master page design and variable data, and this is illustrated in Figure 2(a) which is labeled prior art. Master page **211** is designed, and variable components are merged over master **211**. Two instances of variable data V1 and V2 denoted **213** and **219**, respectively, are shown. The merged result **215** is shown for the case of merging **213** (V1) with master **211** and the merged result **217** also is shown, this the case of merging **219** (V2) with master **211**. A desirable more flexible arrangement is shown in Figure 2(b), which is not prior art. In this second arrangement, the concept of "fixed" and "variable" no longer exists, although it certainly can be accommodated. Any element can be printed as a page (or subpage), or any element can be merged with other elements to create a new page. The order of merging and the mode of merging (overprint, knockout, *etc.*) determines what the final page looks like. In the simple example of Figure 2(b), elements **201**, **203**, and **207** are to be combined, **201** first, then **203** and then **207**. In such a case, the final merged image is the image denoted **205**.

Compression of RIPed Data

One way to overcome the bandwidth problem is to compress the RIPed data before storing and transmitting the data to the printing device. With a compression factor of 10 or higher, the bandwidth becomes manageable with software running on a typical computer system. The problem with raster images is that they contain two kinds of objects with totally different characteristics: 1) line work data ("LW") for text, logos, graphs, block diagrams, *etc.*, and 2) continuous tone ("contone," "CT") data for pictures, blends, *etc.* Because the characteristics of these two types of data are totally different, the typical compression scheme used for these kinds of data is different too. The following table summarizes the characteristics and the compression schemes typically used today:

The flexibility of the Herregods *et al.* system is limited: the system cannot print personalized documents in collated order, as the system cannot deliver the bandwidth required to refresh the master bitmap at the speed of the printing device. Also, master and variable data are combined with special operators in the PostScript file, so object reuse in different jobs is as good as impossible. It would be advantageous to have a method where there is not necessarily a link between the different page elements that make up a final page until actual merging, so different elements can come from different page description files, may be generated using different prepress packages, *etc.*

Merging of Compressed Data

The combination of different page elements into a single page is typically performed by first decompressing the different page elements, combining them and then possibly compressing the complete page again. Since decompression is compute-intensive, this often precludes a pure software solution for the combination performed to send it in real-time to a high speed digital printer. An additional problem with decompressing prior to merging is that the amount of data to be combined increases significantly, placing strain on memory bandwidth which again may preclude a pure software solution. That is, the decompression before combination of the different page elements has the disadvantage that it requires a lot of storage space and a high bandwidth for data transmission.

It would therefore be advantageous to have a method and apparatus for merging different page elements, the page elements being compressed, the merging being carried out substantially without decompressing the data, thus saving computing time as well as storage space. Such an approach is not known in the area of printing. It is known however in another field: motion video, where there is a need to simultaneously receive multiple motion videos from different sources and display them on a single screen via a set of overlaid video windows. U.S. Patent 5,257,113, (October 26, 1993), entitled *VIDEO MIXING TECHNIQUE USING JPEG COMPRESSED DATA*, Chen *et al.*, inventors, describes a method and apparatus for displaying a plurality of video windows (*e.g.*, 30 frames/sec NTSC or 25 frames per second PAL) on a display screen, wherein the data for the windows are mixed in the compressed data domain. All the operations are performed on 8×8 DCT blocks. Chang and Messerschmitt also target broadcasting and video conferencing in their publications on the manipulation of JPEG and MPEG compressed images in the DCT domain (see "Manipulation and compositing of MC-DCT compressed video," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 1, January 1995). In the Chang and Messerschmitt publication, formulas are presented for opaque (*i.e.*, knockout) and semi-transparent (*i.e.*, with an alpha channel) combining of

caching step in the method (and a memory configured as a page element cache in the apparatus), the cache ensuring that page data is available from memory for the merging step.

One embodiment disclosed is a method of generating one or more pages for a digital printing press, the method including the steps of preparing a plurality of page elements, each preferably defined as page description language files. Multipage page description files may optionally be used to generate a series of page elements. Ink properties (knockout, overprint, *etc.*) are included to enable the overlap of different page elements. A technique for defining non-rectangular page elements includes using a transparent white background. Another step is rasterizing (RIPing) and compressing the page elements at the resolution of the output device to form compressed page elements. The compressed page elements are stored in a page element store, which may be implemented in one (or more) of many ways, including as a disk store, a random array of independent disks (RAID) system, a fast optical store, *etc.*

In the method, a merge script (page layout script) is created describing one or more pages by specifying the page elements, and their attributes, including position, margins, overprint order and merge mode (overprint, knockout, overprint based on ink properties, transparency, *etc.*). Theoretically there is no limit on the number or size of elements included on any page.

For each page, the page elements that are included in the page are merged according to the page layout script, the merging generating compressed image data that represents the page, the merging step essentially occurring in the compressed domain. That is, during the merge process, the page elements remain compressed, except that occasionally some boundary pixels may need to be decompressed, when very accurate positioning and arbitrary shaped object masks are included. The compressed image data generated by merging is decompressed into raster image data, and the raster image data printed on the output device (the digital printing press). The steps merging, decompressing and printing are repeated for each page that has to be printed

An aspect of the method is that there is not necessarily a link between the different page elements until the moment that they are brought together according to the merge script, so different elements can come from different page description files, may be generated using different prepress packages, *etc.*

In another aspect, the page elements in the above method include line work data. In yet another aspect, the page elements include CT data. In another aspect, the CT data has associated with it mask data to define a mask around the CT data. In yet another aspect, the page element includes LW data, CT data and mask data. For the latter situation, an embodiment

In another aspect of the method, the line work data in the page elements is at a line work resolution, and the CT data in the page elements is at a CT resolution, the line work resolution being the same as the CT resolution. In yet another aspect, the line work resolution differs from the CT resolution and is at the resolution of the digital printing press.

In another aspect of the invention, the CT compression method is JPEG and uses a set of one or more quantization tables each table corresponding to a compression quality. In a particular embodiment, the tables in the set of quantization tables are related to each other by a set of relationships so that any re-quantization is computationally efficient using one of the relationships.

One aspect of the method of the invention is the ability of merge elements that are defined by arbitrary shapes. In one embodiment, for CT data of a page element, the arbitrary shape is defined by a mask, which may be implemented many ways, including as a bit map and as a clipping path. In one embodiment disclosed, the mask is implemented as a CT validity mask which defines which CT pixels in the CT data of a page element are valid. The CT validity mask is part of the page element. In an embodiment disclosed with a block-based CT compression method, as part of the merge operation, the CT validity mask is used to define boundary blocks and interior blocks of compressed CT page element data. The boundary blocks are merged differently from the interior blocks, and in one specific embodiment, the boundary blocks are merged by first decompressing these to-be-merged boundary blocks.

Another aspect of the invention is an apparatus for merging page elements into compressed merged data. An embodiment of an apparatus is disclosed which comprises a first storage device where page elements are stored in a compressed format. A second storage device is included; a page composition script is stored in it. This page composition script specifies the identifiers, positions and printing order of selected page elements which are to be included in the compressed merged data to be formed. On the basis of what is specified by the page composition script, a merge system retrieves compressed data of the individual page elements stored in the first storage device, and merges them according to the page composition script into compressed merged data, the merging substantially in the compressed domain so that substantially no decompression takes place.

In one preferred embodiment, the first storage device is a RAID system, and the second storage device is a disk drive. In another preferred embodiment, at least one of the storage devices is a memory. The merge system may be implemented as software running on a processor, or a piece of hardware circuitry, which may include a separate processor. for carrying out the function. If the merging means include a processor and a memory, said

The preferred embodiments of the method and apparatus of the present invention have been described assuming, as an example, JPEG compression for CT data and RLE for line work. The method and apparatus disclosed are not restricted to these particular compression formats. They can be used with any compression format as long as the format allows insertion of bitstreams representing compressed image representations into a similarly formatted bitstream. Where these bitstreams should be inserted can then be specified by a marker (when the compression format allows) or a pointer (kept separate from the format).

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a typical computer system in which the present invention may be embodied;

Figure 2(a) illustrates a technique of a master page onto which a variable page (two instances are shown) may be merged. The merged result is the page with variable element V1 merged in one case, and the page with variable element V2 merged in another case; Figure 2(b) illustrates an example of using a flexible technique applied to three page elements which are merged, resulting in a merged image. There is no concept of a fixed master page onto which variable elements are merged, although these concepts may be accommodated;

Figure 3 illustrates a simplified example of two CT page elements to be merged;

Figure 4 illustrates a data stream consisting of compressed block representations each preceded by a restart marker ("RST");

Figures 5(a) and 5(b) show aligning block boundaries of CT images for merging according to one aspect of the invention;

Figures 6(a) and 6(b) illustrate enlarging a CT image prior to merging according to another aspect of the invention;

Figure 7 shows a data stream where only a selection of the compressed block representations is preceded by a marker according to yet another aspect of the invention;

Figure 8 shows an example of run length coded line work data before and after merging according to another aspect of the invention;

Figures 9(a)–(e) show an example of a page element and the structures stored in a page element file according to one embodiment of the invention. Figure 9(a) shows the example of a page element, Figure 9(b) shows the CT data of the element of Figure 9(a), Figure 9(c) shows the CT validity mask for the CT data in the element of Figure 9(a), Figure 9(d) shows the LW

data are then sent to a RIP system as if the designer were using a standard printer. Thus some of these page element pages might be very small—the dimensions of the particular page element.

The overall design of a book is specified by the designer in a page layout script. The structure of the page layout script will be described in more detail later in this description. For database printing where many different instances of some elements act as variable elements, a user designs templates for the layout of the document, and special software is used to generate instances of the actual pages as page elements. This software generates, for the case of "variable" data, a set of pages, each individual page being a particular instance of the page element which takes on the variable data. Note that to the system, there is no notion of "variable data" or a "master" as in prior art systems such as the Barco Graphics PrintStreamer I. In our new system which embodies some aspects of the present invention, all page elements are "equivalent" in the way they are processed. The ordering and merge mode of such elements define how the final page looks.

Note that two RIP systems 1009 are shown connected to LAN 1007 in the system of Figure 10, and more or fewer RIP systems can be used, or the RIP system may be part of workstation 1005, or part of the computer system used to embody the rapid merge system 1003. Each printer driver in the workstation software modules used for design generates an object description in a standard PDL such as Adobe PostScript or Adobe PDF, or the printer driver can output to a proprietary PDL such as HP PCL or GRO. The only requirement in our preferred embodiment is that the PDL enables the RIP (1009) used for RIPing such a PDL to distinguish LW data from CT data. However there is a lot more flexibility in the page design if the PDL also enables the RIP 1009 to distinguish overprinting colors from masking colors, and a masking white color from an empty background, so that the RIP 1009 can encode overprint information in the page elements. Such is the case in the preferred embodiment.

Thus, the designer designs individual page elements as PDL files. In general, these may be single page or multipage PDL files. For example, multipage PDL files preferably are used for "variable" data elements, "variable" in the sense that there is a common design thread that may take on different values in different instances. In such a case, each page may be a different instance. Multipage PDL files are treated as an array of elements, where each element (page) can be accessed individually. Consider, for example, a personalized form letter to be used in a sales promotion. One of the elements of such a design might be the signature of the author of such a letter, and the letter may be used (signed) by one of several salespeople. The signature element might for this example be designed as a multipage PDL file, each page being the size of the largest possible signature to be printed, each page containing the signature of a particular

validity mask can help reduce artifacts of the lossy compression and speed up CT merging by validating certain CT covered by LW. (*e.g.*, CT covered by text is validated; see Figure 9). The CT validity mask can have an arbitrary boundary, it can be made up by disjoint domains that can have any shape. Validating CT under certain LW reduces the validity mask boundary and hence reduces CT merging complexity. Compression artifacts are also reduced because of the fewer boundaries between distinct CT, after CT merging. These otherwise could introduce ringing artifacts due to lossy compression. One simply uses the CT validity mask to indicate which existing CT pixels are not valid. Thus, the CT validity mask is stored as 8*8 blocks of pixels, a special code is used for a block in which all 64 pixels are valid and another code for the case of all 64 blocks invalid. Successive rows of valid or invalid blocks are compressed with run length encoding (RLE). Other blocks are stored as a 8x8 bitmaps. Also, preferably, the CT validity mask is extended beyond the CT selection mask so that it follows the 8*8 JPEG block boundaries of the CT data.

LW data: LW data is compressed with an RLE algorithm, the data header containing a table with the offset to the start of each line in the stream so that, again, the merge system can locate any pixel quickly. The LW resolution preferably but not necessarily is equal to the output printing device resolution, which for the Xeikon is 600 dpi. A *CT selection mask* is a binary mask that enables the decompression system to decide whether an output pixel has to come from the LW data or CT data. The CT selection mask resolution in our system is of the same resolution as the LW data, and is encoded as part of the LW RLE data. Each LW data run thus includes a flag to indicate whether the run is CT or LW (this is the CT selection mask part of the LW data). If LW, the run also includes the color value, and whether the LW data is masking, overprinting, *etc.*

A page element normally would contain both LW and CT data, and in such a case, all three data types (CT data, CT validity mask and LW data, which might include an embedded CT selection mask) are stored together as one file. The system designer would design the file structure to enable efficient printing with the particular printing devices to be connected to the system. A design decision made was to split page elements into horizontal *strips*, each strip having a height of one CT block, *i.e.*, 8 CT pixels. In the Xeikon, since LW is printed at twice the resolution of CT, a strip of LW data thus is 16 pixels high. A subpage in our system consists of a number of consecutive strips, thus at full page width, and with a length chosen so that the amount of memory taken by the subpage can easily be handled by the different buffers in our preferred embodiment. Thus, when a complete page cannot be handled, a subpage is merged at a time. This may happen with such printing devices as the Xeikon, which can print

CT validity mask preferably does not include the area over which the line work "T" occurs in order to reduce the mask's complexity (other implementations certainly may include the T in defining the boundary of the CT). Also note that it is here that the shape of the boundary (circular) is defined. In our implementation, the CT data and CT validity mask are in the CT resolution, 300 dpi, and in Figure 9(d), white are 935 indicates invalid data (mask value 0) and shaded region 939 indicates valid data (mask value =1). LW data 923 includes the LW object (the "T") designed by the user and the background area 925 in transparent white that the RIP included in the LW data. CT selection mask 943 indicates which pixels are CT data and which are LW data, and is used for final combining of the merged CT data and LW data. White regions 945 and 947 indicate that LW data is to be printed (mask value 0) and shaded region 939 indicates CT data is to be printed (mask value =1). The CT selection mask is part of the LW data, and hence has the same resolution, 600 dpi in the case of printing to the Xeikon printing device.

The design decisions discussed herein are implementation details and are not limitations of the invention. Other design decisions may have been made that lead to, for example, different ways of representing boundaries of page elements, different data structures for a page element, different file structures, different ways of handling merging, for example as two-dimensional areas rather than strips, or as complete pages (subpages), *etc.* Thus the implementation details are in no way to be taken as limiting the scope of the present invention.

As discussed further on, there may be situations where the CT validity mask may need to be displaced by at most four pixels. To enable this to happen, in one implementation, the RIP ensures that the objects are larger than the CT selection mask by at least four pixels, and when not the case, enlarges the objects to guarantee this border. In another aspect, when the overall workflow is available, object placement is made so that it always falls on CT block boundaries.

The file comprising all the data for any single page element is then sent to the rapid merge system, denoted 1003, for merging with the other page elements as specified in the page layout script. CT data and LW data are separately merged to generate merged CT and merged LW data, which later is decompressed, combined, and screened. Merging preferably and not necessarily is carried out strip by strip, and separation by separation. A page element is identified by a single file name, and that file includes all the data for that page element. As described earlier, multipage elements also may be used, and are suitable, for example, for describing variable data. These multipage elements can be written sequentially into one file, or they can be written each to a separate file using a file name suffix to identify each individual

intended. With the exception of the input devices and the display, the other components need not be at the same physical location. Thus, for example, portions of the file storage system could be connected via various local-area or wide-area network media, including telephone lines. Similarly, the input devices and display need not be at the same location as the processor, although it is anticipated that the present invention will most often be implemented in the context of personal computers ("PCs") and workstations.

Bus subsystem **115** is shown schematically as a single bus, but a typical system has a number of buses such as a local bus and one or more expansion buses (*e.g.*, ADB, SCSI, ISA, EISA, MCA, NuBus, or PCI), as well as serial and parallel ports. Network connections are usually established through a device such as a network adapter on one of these expansion buses or a modem on a serial port. The computer system may be a desktop system or a portable system or an embedded controller.

Memory subsystem **117** includes a number of memories including a main random access memory ("RAM") **130** and a read only memory ("ROM") **132** in which fixed instructions are stored. In the case of Macintosh-compatible personal computers this would include portions of the operating system; in the case of IBM-compatible personal computers, this would include the BIOS (basic input/output system). In some embodiments, DMA controller **131** may be included. DMA controller **131** enables transfers from or to memory without going through processor **112**.

User input facility **120** typically includes a keyboard **140** and may further include a pointing device **142** and a scanner **143**. The pointing device may be an indirect pointing device such as a mouse, trackball, touchpad, or graphics tablet, or a direct pointing device such as a touchscreen incorporated into the display.

Display subsystem **122** typically includes a display controller **144** and a display device **145** coupled to the controller. The display device may be a cathode ray tube ("CRT"), a flat-panel device such as a liquid crystal display ("LCD"), or a projection device. The display controller provides control signals to the display device and normally includes a display memory (not shown in the figure) for storing the pixels that appear on the display device.

The file storage system **125** provides persistent (non-volatile) storage for program and data files, and typically includes at least one hard disk drive **146** and at least one floppy disk drive ("diskette") **147**. One or more of the disk drives **146** may be in the form of a random array of independent disks ("RAID") system, while others may be more conventional disk drives. The disk drive **146** may include a cache memory subsystem **149** which includes fast memory to speed up transfers to and from the disk drive. There may also be other devices such

buffer of page element cache **1011**, together with some flag or other means of communicating this to the merge system **1015**. When the part of the merge system that reads data from storage seeks such a page element, it then does not need to retrieve it from page element store **1013**, thus saving two disk accesses.

The other aspects of page element cache **1011** deal with retrieval by the merge system **1015** of page elements from page element store **1013**. Such retrieval is made more efficient by the cache **1011** as follows:

- The number of disk seeks of page element storage **1013** is minimized because one reads the page elements one by one into the cache memory **1011**, and then once available in cache **1011**, retrieves page elements rapidly to the merge system **1015** from cache **1011**.
- Page elements that are used multiple times in a book are kept in the page element cache memory **1011**, so that the system need read them only once from the page element storage **1013**.

Note that while cache devices are known in the art, page element cache **1011** differs in that it takes advantage of the fact that we know in advance what is needed. A more detailed description of the element cache **1011** and element store **1013** can be found later in this description.

The Page Layout Script

The page layout script typically is generated in workstation **1005**. In the preferred embodiment, the page layout script is a text (ASCII) description which is an ordered list of books, each book including an ordered list of pages, each page including an ordered list of page elements, and each page element including an ordered list of properties. The properties of any page element include, in order, the element identifier (a number), the element file name (a file name and, for multipage files, a page number within the file of the page element), the position (a horizontal and a vertical position with respect to the top left hand corner of the page), and the merge mode (knockout, overprint, *etc.*). Although one may enter such a script manually using a text editor, one also can use an automatic script generator running on workstation **1005** which is tied to a database system for variable printing. Whether written manually or with the aid of a computer based system, the page layout script is stored in disk storage, which may or may not be the same physical unit as page element storage **1013**. The script is preferably but not necessarily executed by the same processor **112** as that which executes the programming that implement page element cache **1011** and the merge system **1015**. The ordering of the page elements in the ordered list of page elements making up a finished page is important if page

Letter_to_customer). A preamble in the page layout script specifies such details as what units are used for distance measurements. If no units are specified, a default is assumed.

The merge mode indicates how merging will occur. In the preferred implementation, Knockout means the page element knocks out all page elements underneath at that location. All CT page elements presently are merged in Knockout mode, although, as would be clear to those in the art, improved embodiments are possible wherein CT elements may have different levels of transparency, *etc.* using such well known concepts as an alpha channel. For JPEG data, for example, it is known that the DCT is linear, and this linearity can be used to affect merges with alpha channel data. Overprint mode is used for line work only, and states that separation by separation the intensity of the upper object is used, except if it is transparent white. Blackoverprint also only is used for line work, typically text, and states that solid black objects are merged in overprint mode, even if they are generated as knockout objects in the job. Overprint_from_job mode is an overprint mode as specified in the job itself. The particular RLE line work compression method used in the preferred embodiment includes the capability of such knockout specification.

An alternate embodiment to using the above-described ASCII text ordered lists of ordered lists to specify the page layout script is to use a page description script such as PDF by Adobe Systems, Inc. of Mountain View, California. PDF can include external page element references. Because PDF has been used for years and is well documented, it would be clear to one of ordinary skill in the art how to modify the system to incorporate using PDF or other techniques for specifying the page layout script.

Note that no limitations to the scope of the invention are implied by the particular script syntax used in the preferred embodiment. For example, where compressed bitstreams should be merged can alternatively be specified by a marker (when the compression format allows) or a pointer (kept separate from the format) or other means. The computational effort to find such a marker only depends on the specifications of the compression format to allow (aligned or non-aligned) markers. In the case of pointers, a bit pointer indicates the bit from where the bitstream should be merged. If more than one compression format is used to represent a page element, the same method and apparatus can be used for each compression format separately.

The Merge System 1015

The merge system is implemented as a set of computer programs that operate on processor 112. In particular, the computer programs include three threads that execute in

The Decompression and Screening System 1017

The decompression system decompresses the LW and CT data into separate CT and LW pixel buffers. If required, the CT pixels (resolution 300 dpi in our Xeikon based implementation) are scaled up to the device resolution (600 dpi for the Xeikon) by repeating pixels on a line and by repeating lines. The system then selects pixels from one or the other pixel buffer based on the CT selection mask for the merged data. The resulting final raster image is screened, and then forwarded to the print heads of the press 1019. Decompression and screening system 1017 also reduces the data depth to the data depth of the output device. At the final stage, the data rates explode to the full data rate of the printing device 1019. For this reason, in the preferred embodiment, we have chosen to implement the decompression and screening system 1017 as special purpose hardware. Decompression, screening, combining according to the CT selection mask, and buffering the data to the hardware likewise could alternatively be implemented on a small fast computer, or on fast microprocessors or on a DSP device. The present invention does not depend on any particular means (hardware and/or software) for screening or decompression and it is assumed that such functionality is available.

Each particular implementation may be varied to deal with the particular printing devices, and a universal implementation also is possible. One implementation is for the Xeikon printing device. This is a duplex printing device with four print heads (in order of printing, Y C M K) one each side, the print heads separated by 26 cm on each side. Our implementation for the Xeikon system has at least one decompression and screening system 1017 for each color separation for each side of a page.

The Rapid Merge System in Detail

Although not the only way of implementing these aspects of the invention, the rapid merge system in the preferred embodiment is built around a high performance assembler computer such as system 110 of Figure 1, and uses a high capacity storage RAID system as one of the disk drives 146 (or the only disk drive 146) in file storage system 125. It is interfaced for high data rate output via a PCI bus to the special hardware decompression and screening system 1017 and ultimately to a digital printer 1019.

The rapid merge system is now described in more detail with the aid of Figure 12 which shows the operation of the three threads of Figure 11 in more detail. The merging is primarily performed in software by assembler computer system 110 as the three threads, and these threads execute in parallel on processor 112 and cooperate in a pipelined fashion. The flow of operation is controlled by the page layout script, hereinforth denoted by reference numeral 1103, which controls all merging activity. For each page, page layout script 1103

dimension deals with the case when there are several instances of the same page element in any particular page. For example, element number 4 occurs twice in page 4 as indicated by the cross hatches **1311** and **1313**, instance **1311** occurring before instance 1313. Thus the total depth is the maximum number of instances of any page element in any page of the book, and is 3 for the particular instance array **1307** of Figure 13. Clearly, not all entries in instance array **1307** are occupied. In fact, typically, only a few of the entries of instance array **1307** are occupied, *i.e.*, the instance array is sparse. Efficient storage techniques are known for reducing the amount of memory **130** required to store sparse matrices such as instance array **1307**, and such sparse storage techniques are used in the preferred embodiment. Each occupied cell, *i.e.*, each instance in instance array **1307** includes some attributes, and the attributes of each page element instance are shown in Table 2, and include for this instance, the element's position, printing mode, printing order, and the number of the page where the next instance of the element is. This last field refers to the next page after the present page of the instance where the page element is reused. If the present instance is the last instance for the page element, a special value is used for the "next page" field to indicate this fact. The information in this field is used by the caching system to decide which page elements have to remain in the page element cache when a page is merged.

Some attributes of the page elements themselves are stored in memory **130** in a separate one-dimensional array, page element array **1305**. The attributes of a page element stored in element array **1305** are shown in Table 3 and include the element's file name (and if applicable, page within the file) as used in page element store **1013**, a flag that indicates whether the page element is in page element cache **1011**, the address where the page element is stored in the page element cache in the case that it is in the cache (*i.e.*, in the case the flag is set), the page number (or other page identifier) of the page where the next instance of the element is used, and an index of the row in the element instance array 1307 that corresponds to the page element. The attributes of the pages in a book are stored in memory 130 in the form of page description array 1309. The attributes of a particular page in page description array 1309 are shown in Table 4 and include the page's width, height, place in the buffer space (called the output buffer) that includes page buffer(s) **1111**, where the merge result is stored by merge thread **1109** and the index (as an identifying number) of the column of page element instance array **1307** that corresponds to this page.

The first job of read thread 1105 is to pre-fetch page elements from page element store 1013 and put them into page element cache 1011 as necessary. Initially, the first page description pointed to by the read queue 1203 is used, and based on the entry in the page description array 1309 for a particular page, the required page elements to be loaded are determined from the instance array 1307. Required information for loading the page elements into cache 1011 is obtained from the page element array 1305 where, initially, the page elements are referred to by file name. With such information, read thread 1105 knows what page elements to pre-fetch and where they are stored in page element store 1013. Before pre-fetching any page element, the read thread checks whether the page element is already in page element cache 1011 by checking the "Cached" flag in page element array 1305. If this flag is set, no access from page element store 1013 is needed. Otherwise place is allocated in page element cache 1011 and the page element is stored there. After the fetch, the "Cached" flag is set and the "Place in cache" address is written in page element array 1305. Only the first instance of any element is dealt with by read thread 1105 because its job is only to ensure that the elements required by any page are in page element cache 1011. The "depth" dimension of the instance array (the dimension dealing with multiple instances on a page) is not dealt with. Once all page elements in the page description of a particular page are processed, a pointer to the page information entry in the page description array of the completed page is passed to another queue, the merge queue 1205, which also is set up in memory 130 during initialization. This indicates to merge thread 1109 that the elements for that page are available. Merge thread 1109 can then work independently of read thread 1105. Separately, a memory management process described hereinbelow makes sure that elements not yet merged by merge thread 1109 are not removed from the element cache 1011 prior to merging. Thus, when page element cache 1011 becomes full, special action is required as described further on herein.

Cache memory management

Page element cache 1011 comprises a chunk of memory in memory 130. A dynamic memory manager system handles requests from read thread 1105 for a block of memory of a size corresponding to the size of a page element. To prevent cache fragmentation, the page element cache is divided into fixed size chunks (memory pages), and a block thus consists of a number of those chunks. These chunks are not necessarily contiguous in the cache memory. Once the block is allocated, read thread 1105 reads the page element from page element store 1013 into this block When a page element is no longer needed on subsequent pages, as indicated by the appropriate flags in the three array data structures, it is de-allocated by the cache dynamic memory manager under control of the merge thread 1109. Various implementations are possible and may be found in the prior art literature for the dynamic

this, the preferred embodiment page replacement method makes use of the field "Next page used on" of page element array 1305. This field is filled in by read thread 1105 when a page element is read into page element cache 1011, and is copied from the corresponding page element instance field in element instance array 1307. The specific cache replacement strategy now examines this "Next page used on" field and selects all page elements in element array 1305 that have "Cached" flag True and that have "Next page used on" field with pages after the page read thread 1105 is pre-fetching page elements for. All so selected page elements are sorted in descending order of their "Next page used on" field. The preferred cache replacement strategy then uncaches page elements according to the sort order. In this way the optimal cache replacement is achieved.

Merge thread 1109 and the Output Thread(s) 1113

Merge thread 1109 is the core of the rapid merge system 1003. It also runs on processor 112 and executes the implementations of merging methods of the present invention which perform the merging substantially in the compressed raster image domain. The activity of merge thread 1109 is controlled by information referenced in array structures 1307, 1309 and 1305 by page descriptions that are pointed to by merge queue 1205. Using the referenced page descriptions, merge thread 1109 knows where to find the page elements in page element cache 1011 and where to place these page elements on the page. Merge thread 1109 generates all the page separations in compressed format as specified in the set of data structures. It does this separately for CT elements (in the preferred CT compression format JPEG) and the LW elements (in the preferred LW compression format RLE). Typically, for ultimate printing on the Xeikon duplex device, 16 merge results are calculated for each final duplex page: for the front and back sides of the page there are the CMYK separation each in both CT and LW compressed format. The merging results from merge thread 1109 are stored as page buffer(s) 1111 in the output buffer which uses part of memory 130 and which holds the pre-merged pages that are output to the decompression and screening system(s) 1017 by output thread(s) 1113. The number of page buffers and output threads depends on how the data is printed on the particular printing device. Once all the page merge results for a particular page are calculated, the appropriate "Place in output buffer" field in the page description array 1309 is filled in by merge thread 1109 with references to the merge results in page buffer(s) 1111. The pointer to the page description of the page is placed in one or more corresponding queues called the output queues 1209, the queue originally set up in the initialization phase. Each queue indicates to its output thread 1113 that the page data for this page (and this separation in the case of separate page buffers and output threads per separation) as referenced in the page description array 1309 is ready to be processed by the output thread 1113, which outputs the page merge

different separations of different pages at the same time, one page buffer **1111** would store only one separation of the page and thus sometimes is called a page *separation* buffer. In the latter case there are multiple output streams, one for each separation (and print head of the printing device), there also are multiple page separation buffers **1111** coupled to corresponding output queues **1109**. For each output stream a number of page (separation) buffers may be allocated in a memory area of fixed size. In the preferred embodiment, this memory area can store at least two page (separation) buffers **1111** per output queue **1009**. Depending on the compression ratio of the pages, usually a lot more page buffers might fit, and this enables the system to flatten peak loads. The merge thread writes the page (separation) buffers **1111** one by one in the output buffer area, and forwards their reference via the output queue to the output thread. The output thread then forwards the page (separation) buffers to the decompression and screening system **1017**.

Two common methods to put merged page elements into a page buffer are: 1) paint the page elements one by one in the page buffer, according to drawing order; and 2) calculate the paint result of the page buffer scanline per scanline or strip per strip. The second approach often is preferred when merging uncompressed data because memory requirements are high and often a complete buffer for a merged page may not fit into memory. In our case, the data put into page buffer **1111** is in the compressed domain, and the whole of page buffer **1111** (which as discussed earlier is pre-designed to store two pages) remains in memory **130**. Nevertheless, the second approach is preferred. This is because the first approach requires memory management overhead. Contrary to an uncompressed merge, it is not possible to paint (merge) a single page element into proper merged results in-place in the page buffer because the storage size of a compressed page changes as page elements are merged onto it. So either the system must move a lot of data, or it must work with reference pointers to new data at another memory location without being able to release the memory occupied by invalid data. Thus, a lot of memory management overhead would be required. The second approach does not have these problems and therefore is used in the preferred embodiment.

Memory management for page buffer(s) **1111** is now described. The merging result is calculated by merge thread **1109** in strips that correspond to a height of 8 CT pixels (because CT merging preferably is carried out on 8*8 JPEG blocks). The memory for page buffer **1111** is allocated by a page buffer memory manager as strips are merged, according to the maximum size a compressed strip can reach based on a worse case assumption of the lowest compression ratio. The page buffer memory manager typically works in round robin fashion allocating and de-allocating buffers in a reserved memory area which is dimensioned large enough during initialization to hold a chosen number of compressed page buffers **1111** (two in our preferred

merging page elements on a page in compressed form resulting in a single compressed page element, this being the whole page, proceeds according to the following pseudocode, which operates for each strip for each separation:

```
sort the page elements according to their y position on a y-stack,
   topmost element first;
calculate the merge result of each scanline from top to bottom;
      as page elements intersect the current scanline pull them from
         the y-stack and put them on a second stack (an x-stack);
      remove page elements from the x stack that don't intersect with
         the current scanline;
      sort the x-stack according x position, leftmost element first;
      if the x-stack is empty, insert a blank line and continue with
         next scanline;
      pull a page element from the x-stack and put it on a third stack
(a z-stack), call its starting position x;
      if x is positive insert a blank piece till minimum x and width
         of page;
      calculate the merge result of one scanline from left to right;
            pull all other elements from the x stack on the z-stack
               that start on position x;
            call the starting position of the top element on the x-
               stack xn or set xn to the page width if the x-stack is
               empty;
            sort the z-stack according to the page element drawing
               order, lowest element first;
            find the smallest x end position of the elements on the z-
               stack, call it x1;
            for each page element on the z-stack, from lowest to
               highest;
                  get a piece of the page element line that lays on
                     the current scanline, this piece of line goes
                     from x to the minimum of x1 and xn;
                  clip this piece to left or right page border ;
                  merge this piece on the page ;
                  remove the page element from the z-stack and put
                     back on the x-stack when the end of the merged
                     piece is the end of the page element line;
         if the z-stack is empty, insert a blank piece from x1 to xn and
            set x to xn else set x to minimum of x1 and xn.
```

The Merge Process

The merge process performed by merge thread **1109** is now described in detail. Although the operation in the preferred embodiment is merging all the elements at once, strip

CT's, only the CT validity mask comes in effect. After merging one CT E (say 203) on top of another CT D (say 201) through the CT validity mask of D, the CT validity mask is no longer needed. What happens to the CT selection mask after merging two page elements is described in the section on LW merging. The CT selection mask only comes into effect when combining the final merged CT and final merged LW of all the merged page elements. An important distinction between the CT validity mask and the CT selection mask is that the former is at the CT resolution and the latter is at the LW resolution. Furthermore the CT validity mask may validate more of the CT data than is selected by the CT selection mask, but all selected CT data should be validated. An example is shown in Figure 9 where small LW text ("T" 907) is on CT area 909 in the same page element 903. The portion of CT data 911 which is under LW text 927 would be validated, but not selected, as is shown by CT validity mask 933 and CT selection mask 943. This aspect of the present invention enables faster CT merging and reduction of compression artifacts due to less CT border involved in the CT merging process. In the preferred embodiment, CT elements in pixel form (*e.g.*, after RIPing) are compressed using JPEG. That is, CT elements are divided into blocks, each block corresponding to a rectangle, preferably 8 by 8 pixels. In the simplified illustrative example shown in Figure 3, CT page element E, now denoted 301, is 5 by 5 blocks; and CT page element D, now denoted 303, is 2 by 2 blocks. Clearly, actual page elements are much larger. However, when strips are being merged strip-by-strip, each strip is only one block high. The blocks of page element E (301) are numbered E1,1 to E5,5, and those of page element D (303) are numbered D1,1 to D2,2. The convention used is that the first index represents the row number of the block, and the second index represents the column number of the block. It should be clear to those of ordinary skill in the art that in the preferred embodiment, only 1 block-high strips are being merged. The discussion below will in general talk of merging complete two dimensional page elements.

Note also that in the simple case illustrated in Figure 3, the page element D 303 is rectangular, the part of element D 303 to be merged with a part of element E 301 is an exact number of blocks, and that part is merged at a location which in relation to element 301 is at exact block boundaries of page element 301. This is the simplest case. When one knows the complete work flow ahead of time, merging preferably is pre-arranged to guarantee that the merging will be in block boundaries. In another aspect of the present invention, recall that each CT element has an associated CT validity mask and CT selection mask that together define which part of the image is visible. These masks are not necessarily rectangular —they can take any form, for example, a circle. Also, in general, page element 203 can be placed at any arbitrary position 209. Hence, the blocks of page element 201 in general will not align with those of other elements.

As a result of translating D page element 203 with respect to CT validity mask 509, some of the pixels that were outside CT validity mask 509 get inside the CT validity mask and become visible. If CT validity mask 509 is close to or at the edge of page element 203, artifacts may appear after translation. To avoid such artifacts, a border of at least 4 pixels of D page element data must be available around CT validity mask 509 to allow block alignment. In another embodiment of the method of the invention, whenever image CT validity mask 509 of page element 203 is close to the boundary of page element 203, for example, the boundary box of D for a rectangular image, the image is enlarged by a small factor to create a border of 4 pixels around image CT validity mask 509. Such enlarging normally would be done during RIPing as instructions to enlarge the image when both the CT validity mask and the CT data are available. An example of such an enlarging is shown in Figure 6. Figure 6(a) shows image CT validity mask 509 as being the same size as page element 203. The situation after enlargement and before displacement is shown in Figure 6(b). Page element 603 is a slightly enlarged version of page element 203 with a border 605 of 4 pixels outside CT validity mask 509.

In the particular RIP implementation used with the preferred embodiment, the RIP usually has 2 files for each page element: the PS file itself, and a small text file with RIP options (duplex or not, orientation, CT handling, etc.). With such a RIP, the text file can inform the RIP when an object to be RIPed will be used as "variable" data. When the RIP encounters a CT, it examines the current CT validity mask and CT selection mask. The intersection of the bounding box of the CT and the bounding box of the validity mask and clip mask is determined. At each side (top, bottom, left, right) there should be at least 4 CT pixels between this intersection and the CT selection and clip mask bounding box to account for possible translation over a maximum of 4 pixels. If the distance is less than 4 pixels, the CT is scaled and repositioned by the RIP in such a way that a border of 4 pixels is available on all sides. In another embodiment the RIP would not scale the CT but duplicate (or extrapolate) CT pixels beyond the border of the CT validity mask as to ensure that for arbitrarily shaped validity masks a margin of 4 pixels outside the CT validity mask is present.

Sometimes the position of a page element on the final page (relative, for example, to some common page element acting as a "master") is known in advance. If this is the case, the JPEG blocks can be generated in such a way (by the RIP) that they align with the "master," eliminating the need for alignment during the merge phase. If no alignment is needed, the size increase step in the RIP can be omitted.

If the PS file is sent from a standard prepress application program directly to the RIP, using, for example, the standard installed PostScript driver, then one still may incorporate such

boundary of the block. If the intersection is equal to the bounding box of the block, the block is an interior block of D page element 203. If the intersection is not empty and not equal to the block, then the block is a boundary block of D page element 203.

The merging of interior blocks of D 203 "into" page element 201 is now described with the help of Figure 3, "into" used here since the order is that D is over E Figure 3 for simplicity assumes that there are no boundary blocks in the D image, shown here as 303. The underneath page element in this simple example is shown as 301, and the desired merged image, as 305. In this embodiment, it is assumed that these CT images are compressed using JPEG, as described above. M image 305 is constructed by copying those blocks (compressed) from E page element 301 which need to be retained in M image 305, and all the interior blocks (compressed) of D page element 303, which in this simple case, are all the blocks of D page element 303.

The following notation is used in describing how one locates the boundary blocks to copy:

- (Px, Py) are the vertical and horizontal positions, respectively, in block coordinates of the location 307 where page element D should be printed (in knockout) over page element E. The example of Figure 3 shows page element D (303) to be placed over page element E (301) to give a combined image 305, (Px, Py)=(3, 2);
- Ei,j is the designation of a block in page element E, with $1 \leq i \leq$ E_max_vert and $1 \leq j \leq$ E_max_hor, where E_max_vert and E_max_hor are the maximum block designations in the vertical and horizontal directions respectively, for page element E; and
- Di,j is the designation of a block in page element D, with $1 \leq i \leq$ D_max_vert and $1 \leq j \leq$ D_max_hor where D_max_vert and D_max_hor are the maximum block designations in the vertical and horizontal directions respectively, for page element D.

The block numbers of the source blocks to be copied are determined by the following pseudocode, which is shown for a two dimensional object. As would be clear to one of ordinary skill in the art, the "`For i = 1 to E_max_vert`" and "`next i`" statements may be removed from the pseudocode when dealing only with a one block high strip of CT data.

In the preferred embodiment, 203 and 207 represent different CT page elements to be included in the final merged image 205. The final merged image is in the form of compressed merged CT data and may represent a complete page or part of a page.

When 203 and 207 represent different CT page elements to be included in the final merged image, the CT selection masks associated with each of these CT data also need to be merged. These are merged at the same time as the LW data is merged as described later herein.

Although the example used above to illustrate copying the interior blocks of D uses rectangular images, the method according to the present invention is not restricted to rectangular images. As discussed above, in general a mask 509 can accompany a non-rectangular page element D 203, indicating whether a block is a valid block of page element D or not. In this case, the (2-D copying) pseudocode above is applied only to interior blocks of page element D(203).

For the illustrative example of Figure 3 or interior blocks of D (203), it has been assumed that in case of the JPEG compression format, the E and D images are compressed using the same JPEG scheme, including the same quantization tables and Huffman tables. When this is not the case, when later decompressing the merged data, the decompression scheme needs to be switched to the particular one of the copied block when such a block is encountered. How this is achieved is as follows. The compression module in our particular RIP knows only one Huffman table, and a limited set of quantization tables. This is not a limitation of the invention, simply an implementation choice. These quantization tables allow for different quality settings (high, medium or low). In a configuration having a "master" and a "variable" object, if a master is compressed with a high quality table and a variable object with a medium quality table, then the variable must be re-quantized. In a general case this would introduce additional rounding errors and thus an additional quality loss. To minimize this, the particular embodiment uses a set of different quantization tables such that the quantization coefficients of the medium quality table are the double the corresponding coefficients for the high quality table. The coefficients for the low quality table are then again the double. The merge thread 1109 re-codes all the page elements to the highest quality setting used on a page.

In the preferred embodiment, boundary blocks of D page element 203 are not merged in compressed form since each resulting block in merged image 205 contains a mix of pixels from the respective block in E page element 201 and the boundary block of D page element 203. So boundary blocks in D and the respective block in E are decompressed, merged in uncompressed form, and re-compressed to form the respective block in M image 205. It should be noted that the merging of boundary blocks using traditional uncompressed methods has

is, the superblocks are rows. For example, considering page element D (303) of Figure 3, the blocks D1,1 and D1,2 form one row; and the blocks D2,1 and D2,2 form a second row. The data stream of page element D is now modified by including a first RST marker before block D1,1 and a second RST marker before block D2,1. RST markers are spread over page element E only when a new row of blocks starts. With such a scheme, it often is necessary to search for a block which is not preceded by a marker. In this embodiment, search for such a block requires *partial* decompression, and only of the data stream following the RST marker just before the location where D blocks should be placed (in knockout). Only partial decompression is needed because the actual pixel values are not required—only block boundaries are. In the case of JPEG compression, this partial decompression could be just the Huffman decoding step of JPEG decompression, resulting in a fixed length data stream for each block. This way, one can use the RST markers to skip to the location, but before the block position where replacement starts, and from then on one performs only partial decompression as far as is needed to reach the block position where replacement starts, which takes much less time than complete decompression. This embodiment thus involves a combination of compressed merging and partially compressed merging. Using this embodiment, the number of required RST markers is reduced, increasing the compression ratio, but at the cost of introducing additional time needed for the partial decompression.

Merging of Line Work Data (including the CT selection Mask)

Another feature of the present invention is for the case of page element 201 and page element 203 both comprising line work data. Line work data herein includes the CT selection mask as well as line work as is commonly understood in graphic arts, including logos, text, and other forms characterized by sharp boundaries. In the preferred embodiments, it is assumed that line work is represented by raster data, that is, in the pixel domain. This is the case, for example, if all page elements are available post RIPing, and preferably pre final screening, a RIPed image in this case comprising pixels, each pixel having four color values, C, M, Y, and K. As with CT data, LW data is merged one separation at a time.

In the preferred embodiment, the RLE used to compress line work is characterized as follows:

- The raster data is arranged row by row, column by column.
- For each row (or column, in some embodiments), adjacent pixels with equal color are grouped into *runs*. A run is a data structure defining the color and the number of pixels in the run of that color, that number called the "run_length".

The merging step for line work data is now described. In the preferred embodiment, merging operates row by row, so the merging of only one row will be described with the aid of Figure 8. In Figure 8, only part of the E page element row, denoted **803**, is shown. This is the part that contains those runs which either partially or totally overlap with the corresponding row of page element D, that row shown as **805**. The four runs of row **803** partially or totally overlapping with **805** are denoted **809**, **811**, **813**, and **815** respectively. All runs of page element row **805** are shown. Runs **817**, **821** and **829** are in the background color, so are not visible in the merged image, while runs **819**, **823**, **825**, and **827**, shown hatched, are to be merged into row **803** of E. Assume that the row **805** of page element D (**203**) is to be merged at horizontal position X relative to the start of row **801** of page element E (**201**). The resulting merged row is shown as row **807** in Figure 8. In the preferred embodiment, the merging operation comprises the following steps:

Starting from the left,

- Runs of row 803 of E before the first row 809 of E that partially or totally overlap with the row 805 of D are copied unchanged to the output.

- Runs of row 803 of E that partially or totally overlap with row 805 of D are copied to a temporary storage area as an ordered list. In Figure 8, these copied rows are rows **809**, **811**, **813**, and **815**. Each run length in the elements of the ordered list is replaced by the x-coordinate of the first pixel in that run. Referring to Figure 8, the x-coordinate of the first run **811** is obtained as described above using an index i. Coordinates of subsequent runs are obtained by adding run lengths to the x-coordinate of the first run **811** as additional runs are copied. Thus a temporary list is determined, the list including coordinate x_1, at the color of run **809**, coordinate x_3 at the color run of **811** , coordinate x_6 at the color of run **813**, and coordinate x_{10} at the color of run **815**. An extra element is now added to the end of the list with the coordinate entry the coordinate of the last pixel of the last run of E, **815**, plus 1, and a color "don't care". In Figure 8, the x-coordinate of the additional item is $x_{13}+1$.

- The row runs of D are copied to a second temporary storage area as a second ordered list, and again, the run lengths in each list element are replaced by the x-coordinate of the start of that run. The first element as x-coordinate $X(= x_2)$. Again, an extra item is added to the list, the x-coordinate of the extra item being the coordinate of the last pixel of the last run in page element-row **805**, plus one, and the color of the extra element being *background*. For the situation depicted in Figure 8, the linked list comprises (color of **817**, x_3), (color of **819**, x_4), (color of **821**, x_5), (color of **823**, x_7), (color of **825**, x_8), (color of **827**, x_9), (color of **829**, x_{11}), (***background***, $x_{12}+1$).

In the example of Figure 8, the runs starting as x_2, x_{10}, and $x_{12}+1$ are deleted. The background color of the run starting at x_5, is replaced by the color of **811**, and the background color of the run starting at x_{11} is replaced by the color of run **815**.

The last run in the list (the run starting at $x_{13}+1$) is now deleted. For the example, the list now includes the following runs: (color of **809**, x_1), (color of **811**, x_3), (color of **819**, x_4), (color of **811**, x_5), (color of **813**, x_6), (color of **823**, x_7), (color of **825**, x_8), (color of **827**, x_9), (color of **815**, x_{11}).

At this stage, the x-positions in the list are replaced by the run-lengths; these run lengths determined from successive x-values. Also, any zero-length runs are deleted from the list, and the result is copied as the merged row **807**.

The remaining runs of the row of page element **201** are now copied to the output.

Thus the line is merged. This is carried out for all lines. It should be noted that in all the above steps, decompression never takes place. Thus, a method of merging compressed line work images without decompressing has been described.

In another embodiment, transparent overprinting is made possible by appending the run information, being run length and run color, with run alpha. The run alpha is a measure for transparency of the run. When merging two pieces of runs of the same length, one run piece with run color "a" and run alpha "t" on top of another run piece with color b, then the resulting run color is taken to be min((a*t+b*(max_alpha-t))/max_alpha, max_color). Where "max_alpha" is the maximum value of the run alpha and max_color is the maximum color intensity value. In yet another embodiment the color of the resulting run is taken to be just max(a, b). In this context, colors (CMYK) of increasing intensity have increasing color values *i.e.*, for white all color separation values are zero.

When more than one page element is to be merged with page element **201**, for example, there is also an object **207** in addition to element **203**, the compressed merging process can be performed, just as in the CT case, by first merging 201 and 203 into an intermediate merged image. and then merging 209 into the intermediate merged image.

Alternatives to Speeding Up the Copying Operations

In some applications, a large number of customized merged pages may need to be generated based on the same E page element. In such a situation, the same blocks need to be located many times. The page element cache 1011 already has been described.

pair. When the copy operation is done, DMA controller 131 interrupts the processor 112. In this way processor 112 is involved only for sending each (address, count)-pair to DMA controller 131. In one embodiment, DMA controller 131 has scatter-gather functionality, which means that it is able to access a list in memory containing several, possibly many (address,count)-pairs. DMA controller 131 then executes the copy operations for each of these pairs automatically. In this case, processor 112 is involved even less at the startup for creating a list of pairs and forwarding the start address of the list to DMA controller 131 and at the termination of the last pair of the list.

In another embodiment of this invention, (address, count)-pairs representing the location and amount of data that needs to be copied from either E or D, as the case may be, are set up and loaded into memory, the start address of the first pair is loaded into DMA controller 131 with scatter-gather capability, and the DMA transfer is commenced. As the copying process occurs using special purpose hardware, the merging process can be significantly speeded up.

Apparatus for Merging

A schematic illustration of an apparatus for merging page elements into compressed merged data according to an aspect of the present invention is shown in Figure 14. The apparatus comprises a first storage device 1401 where images, preferably raster images of individual page elements or groups of page elements, are stored in a compressed format. In a second storage device 1402 a page composition script is stored. This page composition script specifies the identifiers and the positions of the page elements or groups of page elements which are to be included on the compressed merged data to be formed. The compressed merged data may represent a complete page or part of a page. On the basis of what is specified by the page composition script (also called the page merge script and page layout script), merging means 1403 retrieve compressed data of the individual page elements or the groups of page elements stored in the first storage device 1401, and merge them according to the page composition script into compressed merged data. No decompression has been carried out in order to obtain the compressed merged data.

Storage devices 1401 and 1402 may be part of the file storage system 125 of assembler computer system 110 of Figure 1. In the preferred embodiment, storage device 1401 is a RAID subsystem in file system 125, and storage device 1402 is a disk drive. Alternatively, storage device 1401 may be a large high speed buffer memory device comprising a large amount of RAM for storing the raster images of individual page elements or groups of page elements in a compressed format, so that this data is rapidly available for processing.

CLAIMS:

1. A method for merging a first page element (201) with a second page element (203) into a merged page (205) for printing, the merging according to a pre-specified order and positioning, the first and second page elements (201, 203) stored in a page element store (1013) in a format, said format for each page element including compressed raster image data of the page element, the method comprising the steps of:

   (a) reading the compressed raster image data of the first and second page elements (201, 203) from the page element store (1013) into a memory;

   (b) merging the compressed raster image data of the first page element with the compressed raster image data of the second page element to form compressed merged raster image data, the compressed merged raster image data describing the printed appearance of the merged page, the merging according to the pre-specified order and relative positioning, the merging substantially without decompressing the compressed raster image data of the page elements;

   (c) decompressing the compressed merged raster image data into printing data; and

   (d) printing the printing data.

2. A method of generating a page for an output printing device (1019), the method including the steps of

   (a) preparing a plurality of page elements to be combined on the page, each page element in the form of a corresponding page element description;

   (b) RIPing and compressing the page element description of the page elements to form compressed rasterized page elements;

   (c) storing the compressed rasterized page elements in a page element store (1013);

   (d) specifying the positioning and order of merging and type of merging for the page elements on the page, the specifying generating a page layout script (1103);

the CT selection mask (943) identifying each pixel in the page element as a line work pixel or a CT pixel.

8. The method according to claim 7 wherein the CT selection mask is in compressed form in the compressed rasterized page element determined from the page element description of the CT data-containing page element.

9. The method according to claim 6 wherein said step (e) of merging further includes

(i) merging the mask data for all page elements that include CT data to form compressed mask data

(ii) merging the compressed line work data of all the page elements that include line work data to form compressed merged line work data, and

(iii) merging the compressed CT data of all the page elements that include CT data to form compressed merged CT data,

the merging of mask data, compressed line work data and compressed CT data occurring substantially in the compressed image domain.

10. The method according to claim 9 wherein the step (f) of decompressing comprises:

(i) decompressing the compressed merged CT data to form decompressed merged CT data,

(ii) decompressing the compressed merged line work data to form decompressed merged line work data, and

(iii) combining the decompressed merged line work data and the decompressed merged CT data to form the raster image data.

11. The method according to claim 7,

wherein the step (e) of merging further includes

(i) merging the mask data for all page elements that include CT data to form a merged CT selection mask,

the compressed line work data to enable quickly locating a desired location in the compressed line work data, the line work location data including a lookup table containing an offset to the beginning of each row of line work data.

17. The method according to claim 14 wherein the compressing of the CT data in said step (b) of RIPing and compressing includes adding CT location data to the compressed CT data to enable quickly locating a desired location in the compressed CT data, the CT location data including restart markers in the JPEG data forming the compressed CT data.

18. The method according to claim 13 wherein the relative positioning of CT data to be merged is on the boundaries of the blocks defined by the CT compression method.

19. The method according to claim 18 wherein step (e)(iii) of merging the CT data includes repositioning the CT data so that merging occurs on CT block boundaries.

20. The method according to claim 6 wherein the line work data is at a line work resolution and the CT data is at a CT resolution, the line work resolution being the same as the CT resolution.

21. The method according to claim 6 wherein the line work data is at a line work resolution and the CT data is at a CT resolution, the line work resolution being different from the CT resolution and being at the resolution of the output printing device.

22. The method according to claim 14 wherein the JPEG method uses a set of one or more quantization tables, each quantization table corresponding to a corresponding compression quality.

23. The method according to claim 22 wherein the set of quantization tables includes a plurality of compensation tables that are related to each other by a set of relationships designed for making re-quantization computationally efficient.

retrieve the compressed data when run on one of the processors, and wherein the merging means includes merging programming instructions which merge the compressed data when run on one of the processors.

30. The apparatus for merging according to claim 29 further comprising:
(d) decompressing means (1404) for decompressing the compressed merged data into printing data for printing on an output printing device (1019).

31. The apparatus for merging according to claim 27 wherein one of the plurality of page elements includes line work data compressed using a line work compression method and wherein one of the plurality of page elements includes CT data compressed using a CT compression method.

32. A system for merging page elements onto a page to form merged page data and for printing the merged data on an output printing device (1019), the output printing device including print heads operating at a print rate, the system including

(a) a RIP (1009) to form compressed rasterized page element data from page element data;

(b) a page element store (1013) for storing the compressed rasterized page element data;

(c) a merge system (1015) for running a page layout script (1103), the page layout script (1103) specifying the positioning and order of merging and type of merging for the page elements on each of a set of pages, the merge system (1015) including:

(i) a read subsystem for retrieving compressed rasterized page element data from the page element store according to the page layout script, and

(ii) a merger for merging the retrieved compressed data according to the page layout script, the merging substantially in the compressed domain and forming merged compressed data; and

(d) a decompressor to form decompressed page data from the merged compressed data.

using data from its associated input thread and loading data into its associated output thread.

41. The system according to claim 34 wherein the decompressor decompresses the line work merged compressed data into decompressed line work page data and the CT merged compressed data into decompressed CT page data.

42. The system according to claim 41 further comprising

(e) a combiner which combines the decompressed line work page data and the decompressed CT page data into printing data and feeds the printing data to the print heads of the output printing device at the print rate required by the print heads.

43. The system according to claim 37 wherein the decompressor decompresses the line work merged compressed data into decompressed line work page data and the CT merged compressed data into decompressed CT page data, the system further including a combiner/screener which combines the decompressed line work page data and the decompressed CT page data into page data, screens the page data and feeds the screened page data to the print heads of the output printing device at the print rate required by the print heads.

44. The system according to claim 34 wherein the line work compression scheme is a run length encoding scheme and the CT compression scheme is a JPEG compression scheme.

45. The system according to claim 39 wherein the read system comprises a set of programming instructions that when loaded into a processor memory of a processor and run on the processor, carry out a set of steps including, for each page specified in the page layout script,

- for each page element of the specified page, checking if the page element is already present in the page element cache and loading into the page element cache each page element that is not yet present in the page element cache, the loading comprising:

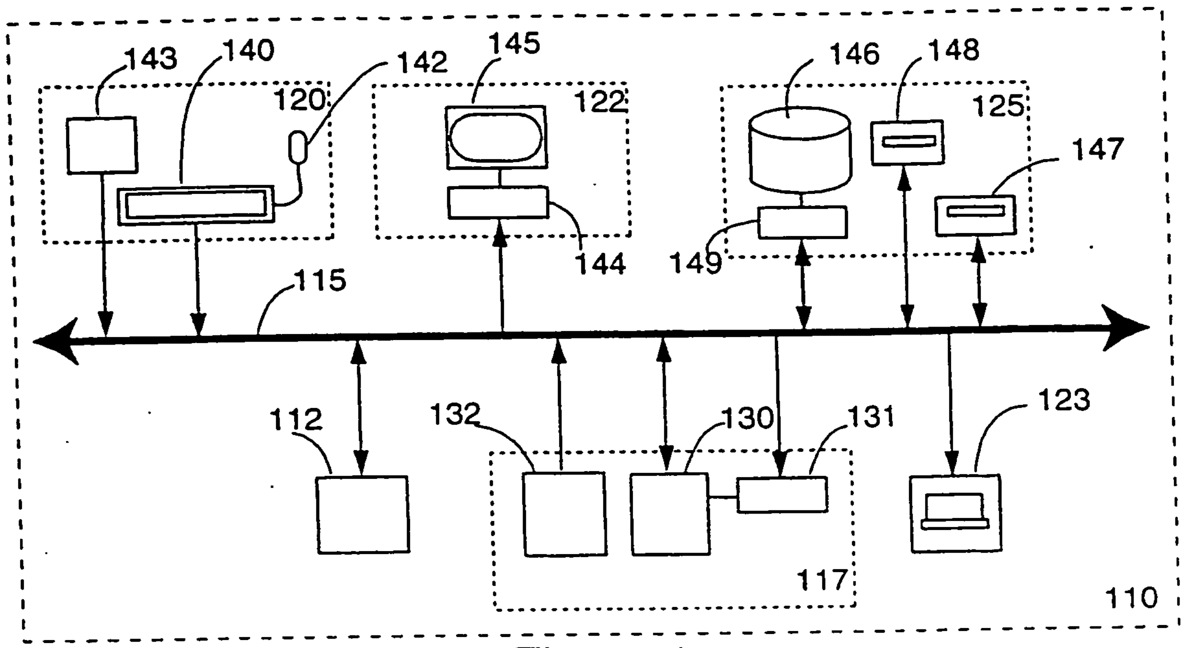


Figure 1

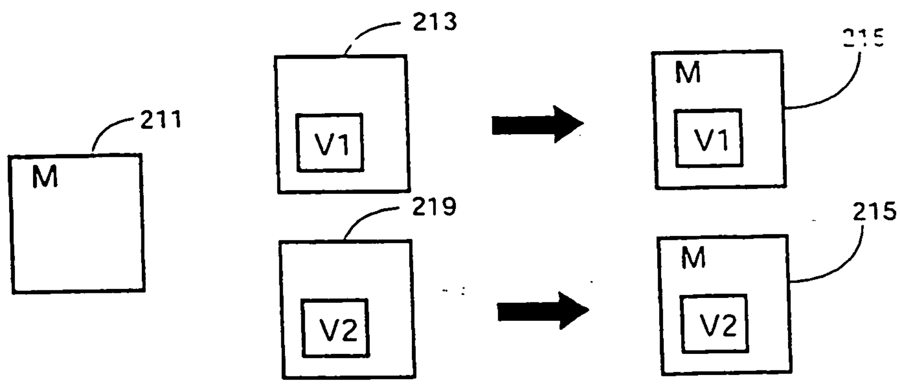


Figure 2(a) (Prior Art)

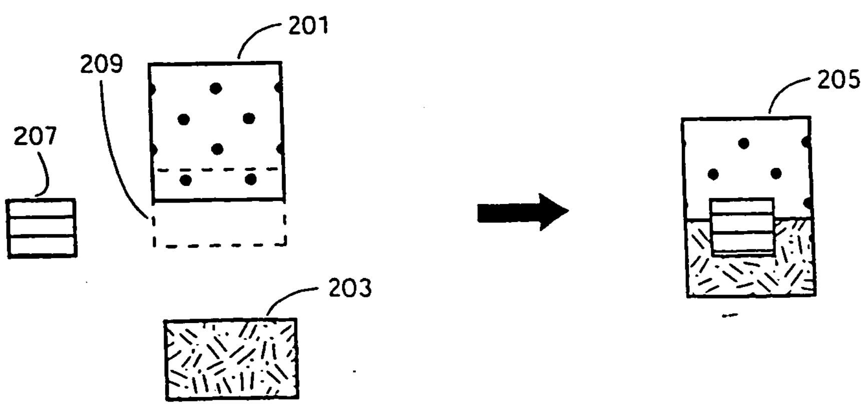
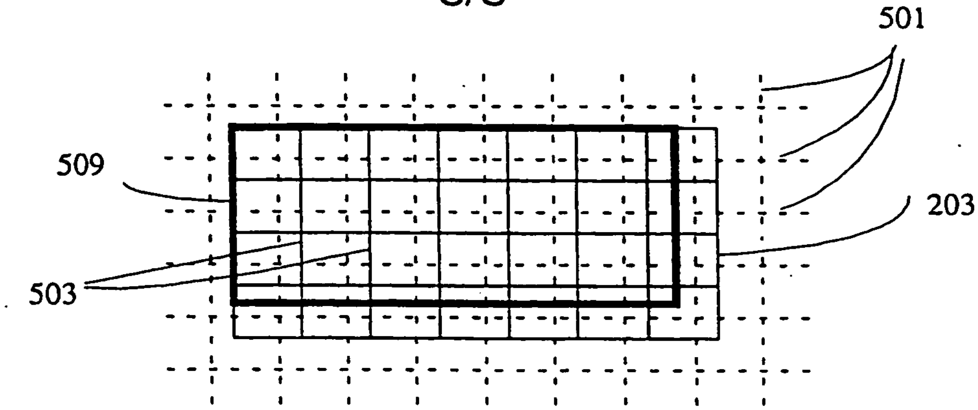
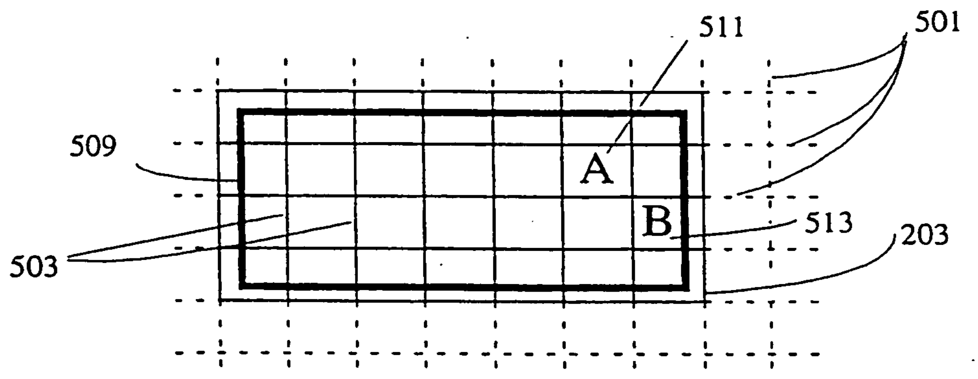


Figure 2(b)

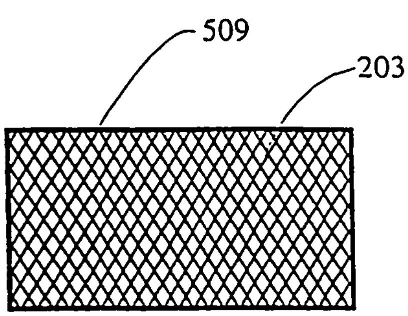


(a)

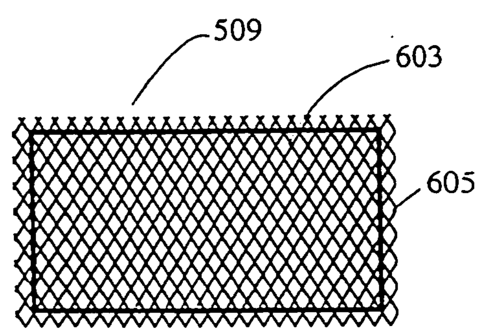


(b)

Figure 5



(a)



(b)

Figure 6

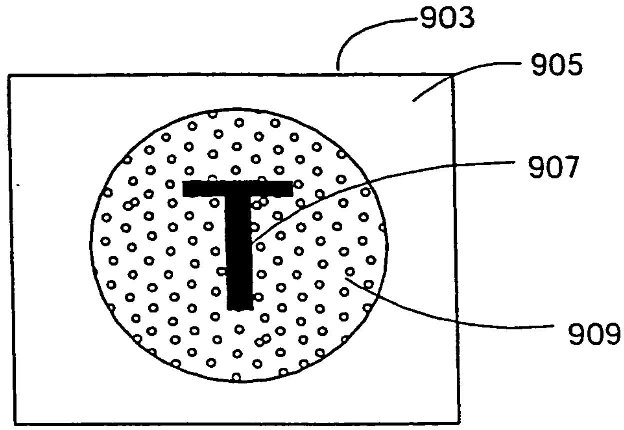


Figure 9(a)

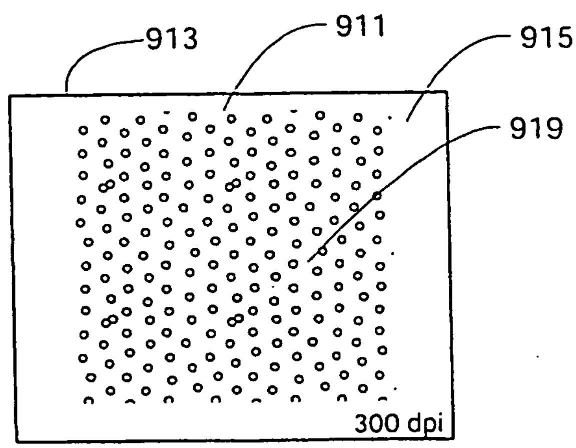


Figure 9(b)

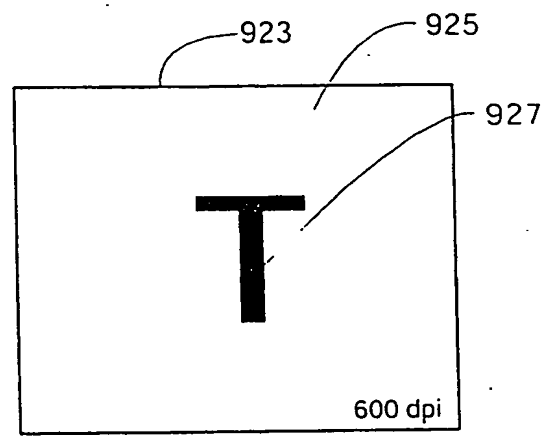
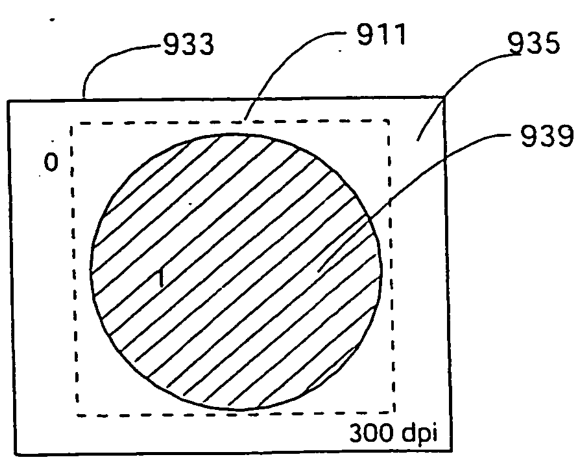
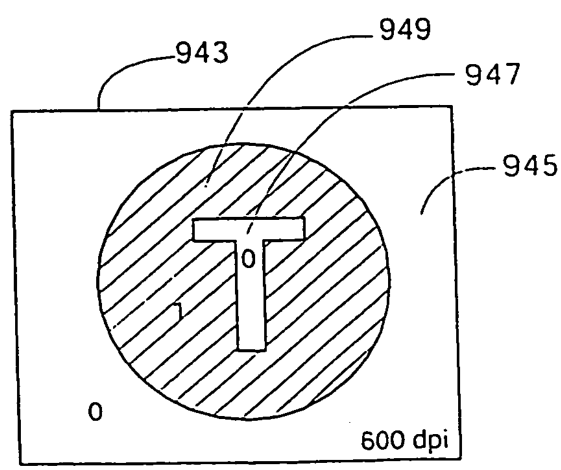


Figure 9(c)



0 = transparent CT
1 = masking CT

Figure 9(d)



0 = LW
1 = CT

Figure 9(e)

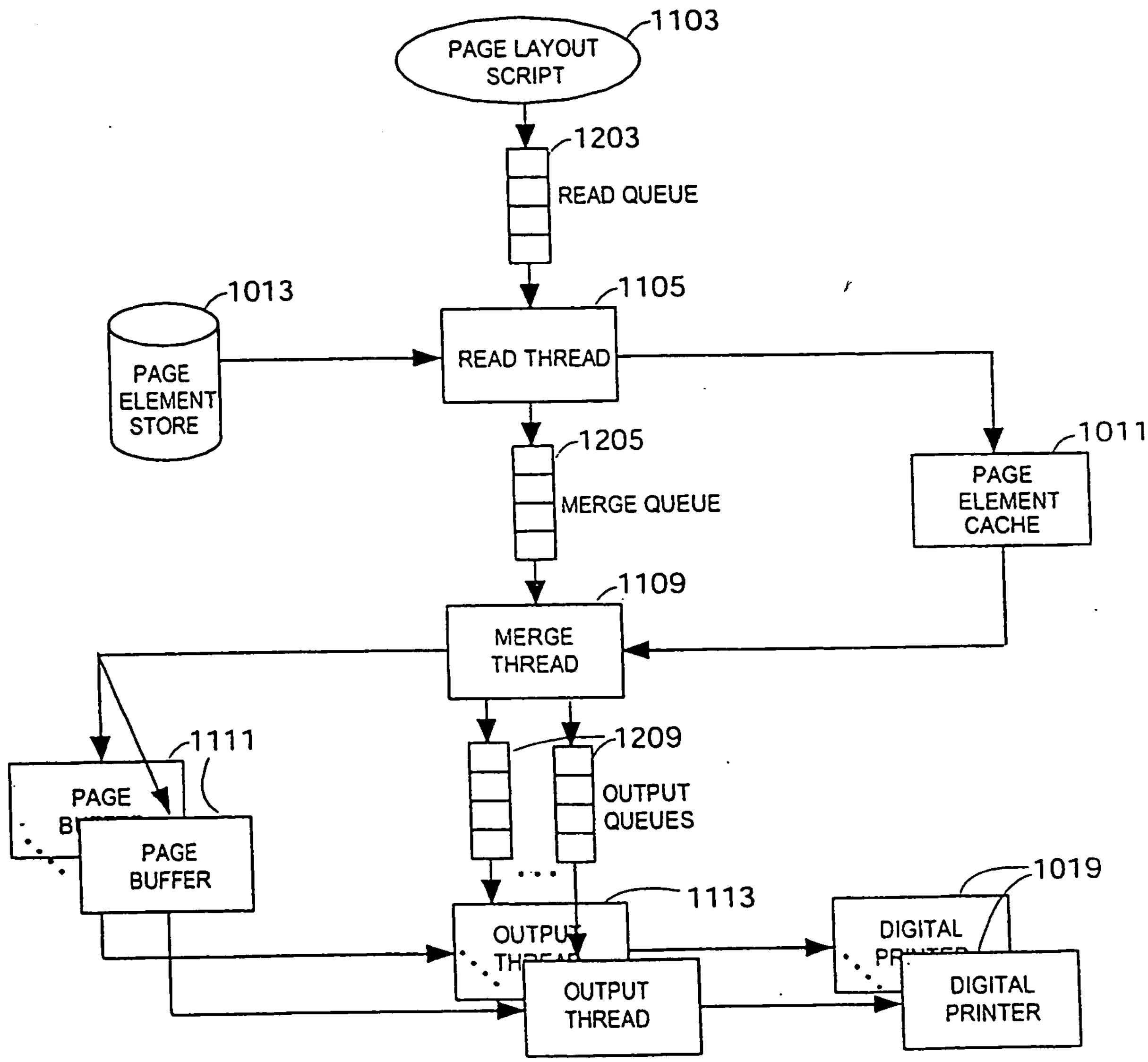

PAGE LAYOUT SCRIPT
1103
1203
READ QUEUE
1013
PAGE ELEMENT STORE
READ THREAD
1105
1205
MERGE QUEUE
PAGE ELEMENT CACHE
1011
MERGE THREAD
1109
1111
PAGE BUFFER
PAGE BUFFER
1209
OUTPUT QUEUES
1113
OUTPUT THREAD
OUTPUT THREAD
1019
DIGITAL PRINTER
DIGITAL PRINTER

Figure 12

INTERNATIONAL SEARCH REPORT

International Application No
PCT/BE 98/00169

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06K15/02 G06F3/12

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06K G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X A	EP 0 570 146 A (XEROX CORP) 18 November 1993 see column 3, line 2 - column 4, line 13 see column 12, line 8 - line 37 ---	1 2,27,32
A	US 4 493 049 A (LING ANDREW T ET AL) 8 January 1985 see column 3, line 56 - column 4, line 68; claims ---	1,2,27, 32
A	EP 0 473 340 A (CANON INFORMATION SYST RES ;CANON KK (JP)) 4 March 1992 see abstract; claims see column 10, line 55 - column 14, line 17 ---	1,2,27, 32
A	EP 0 613 102 A (ADOBE SYSTEMS INC) 31 August 1994 ---	

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
8 March 1999	12/03/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Gélébart, Y

1